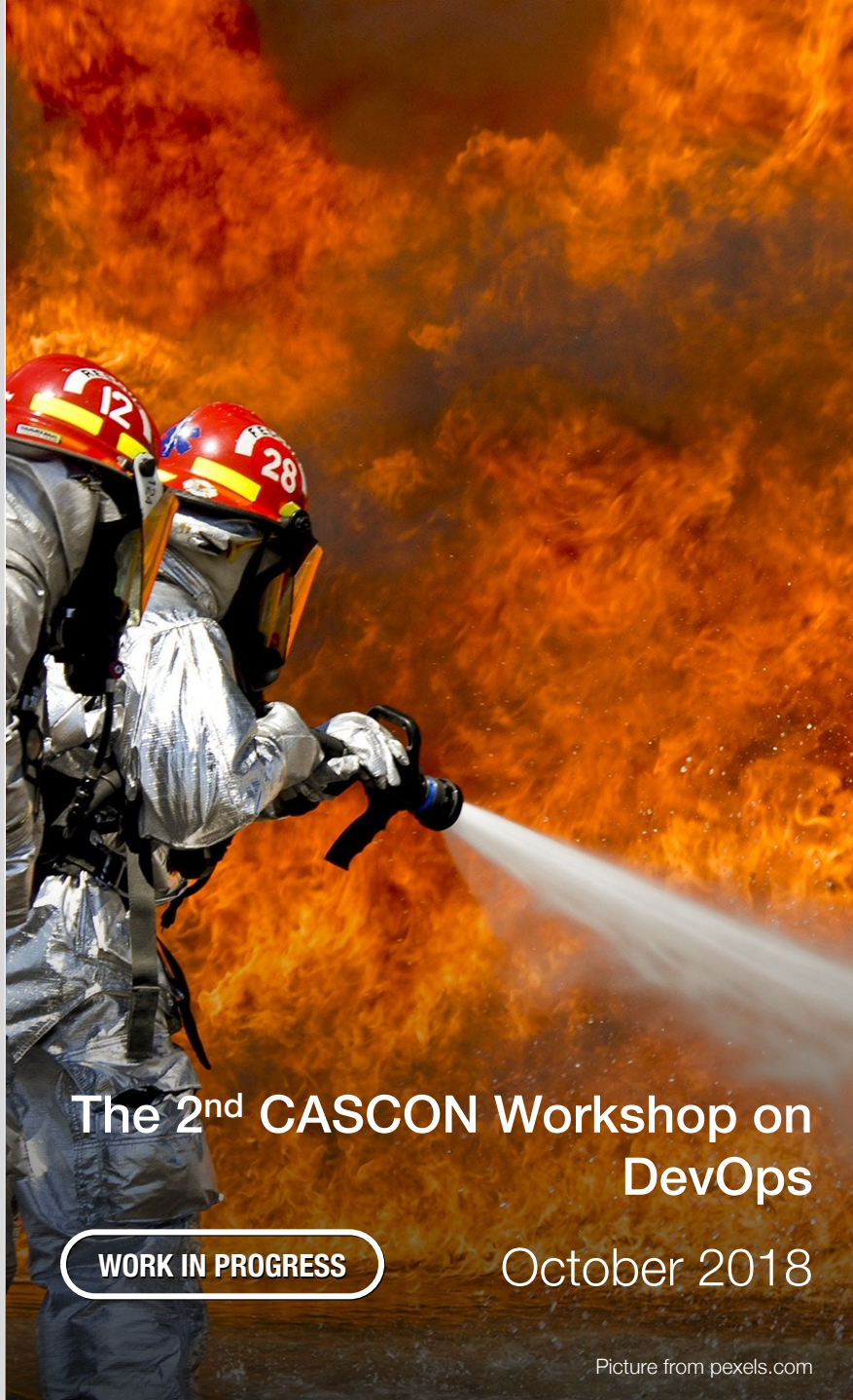
Towards Continuous Assurance of Non-Functional Requirements Through Continuous Experimentation

Miguel Jiménez, Hausi Müller
{miguel, hausi}@uvic.ca

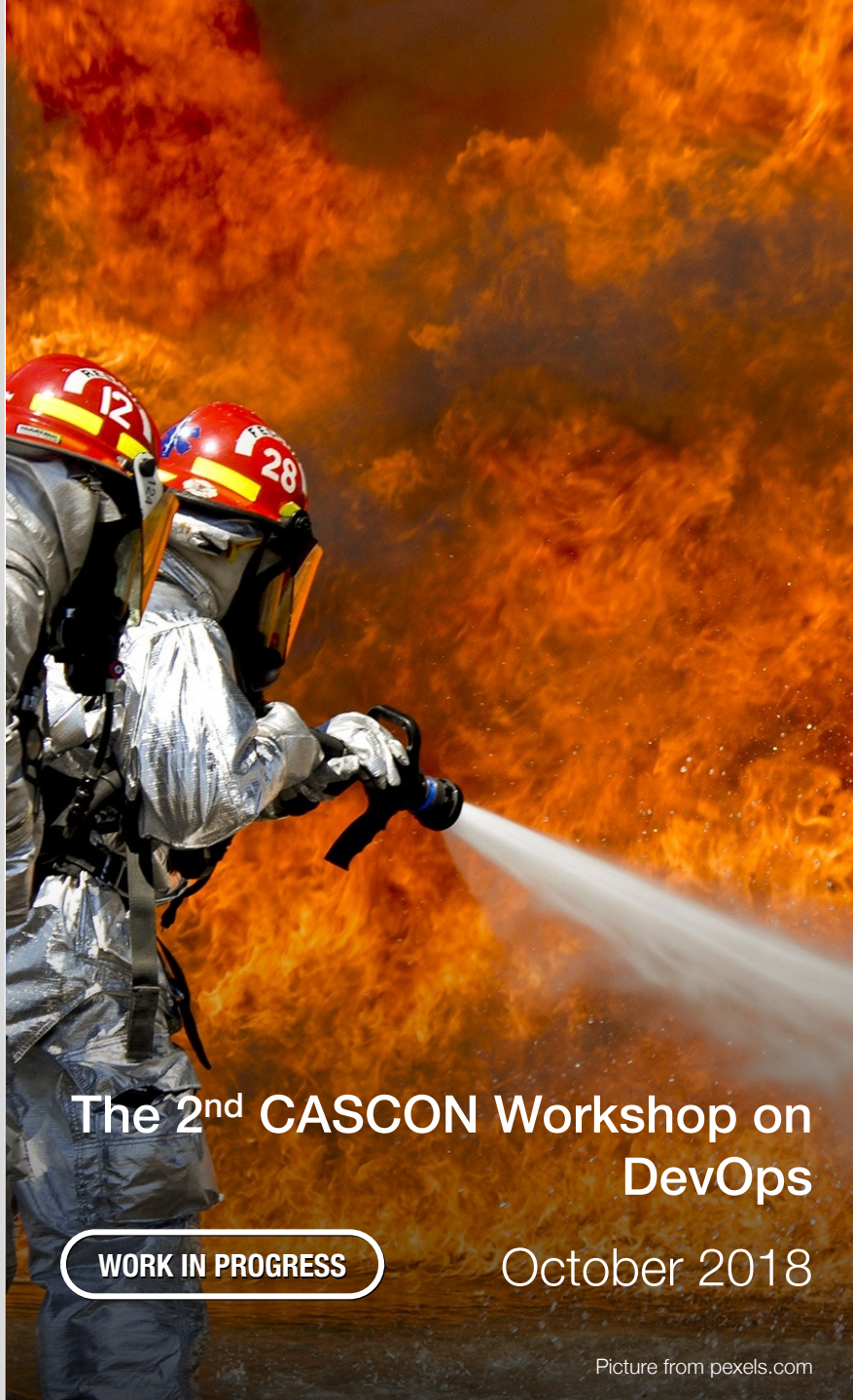
Gabriel Tamura
gtamura@icesi.edu.co



Towards Continuous Assurance of **Operationalizable** Non-functional Requirements Through Continuous Experimentation **at Run-time**

Miguel Jiménez, Hausi Müller
{miguel, hausi}@uvic.ca

Gabriel Tamura
gtamura@icesi.edu.co




12
09
TUE, MAY 9
98%

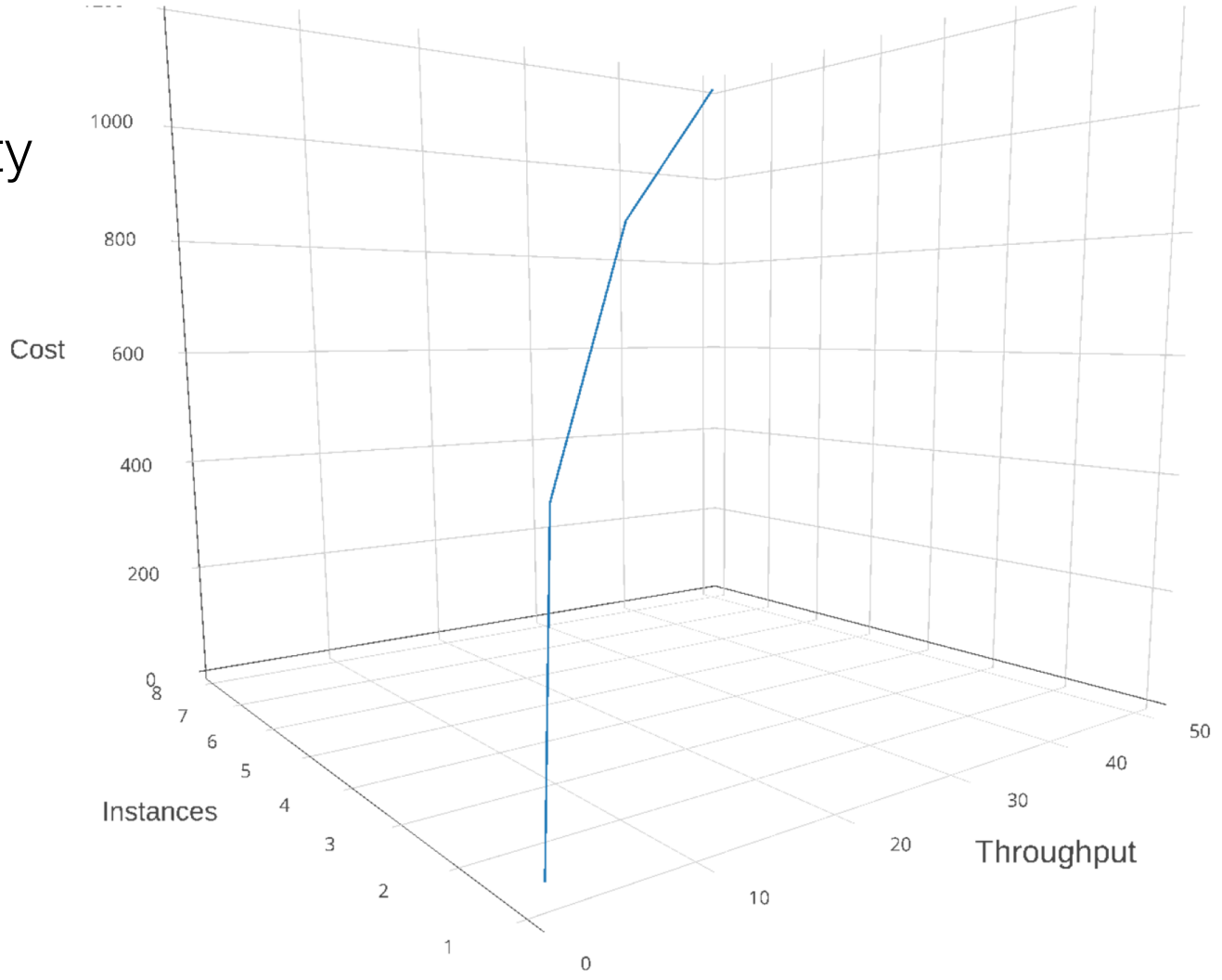
DEPUTY
CHIEF

Picture from pexels.com

Picture from pexels.com

Common Assurance Practices

- Microservices: scalability & availability
- Implementation-specific strategies, e.g., granular bandwidth control
- Non-functional requirements are addressed during Dev and Ops
- Traditional SE practices leave out uncertainty management



→ run-time changes require continuous assurance

Continuous Experimentation

Not used enough

- Common practices: business- and regression-driven experiments
- Deployment practices are crucial in carrying out experiments
- Prototypes for functional requirements, experimentation for the non-functional ones

Experimentation for Non-Functional Requirements

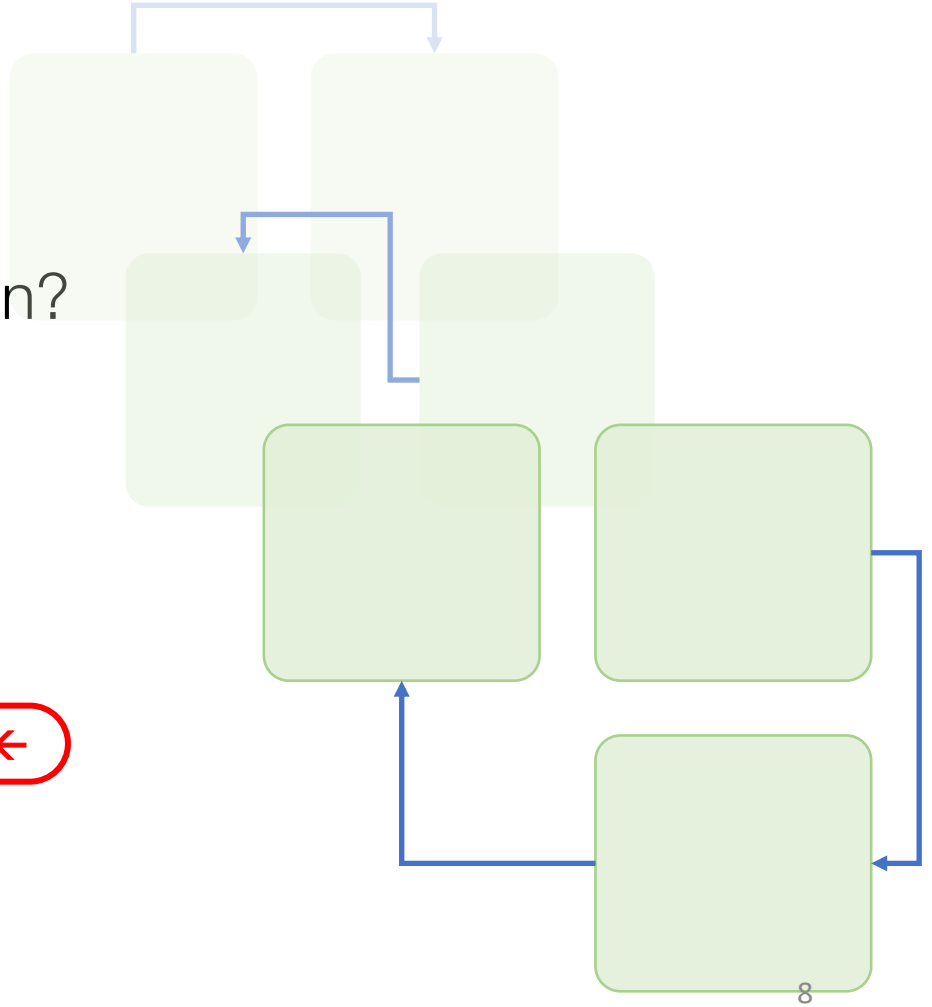
DESIGN-TIME

- Evaluation of design trade-offs
  - How to conduct eval. without implementation?
- Incremental design

RUN-TIME

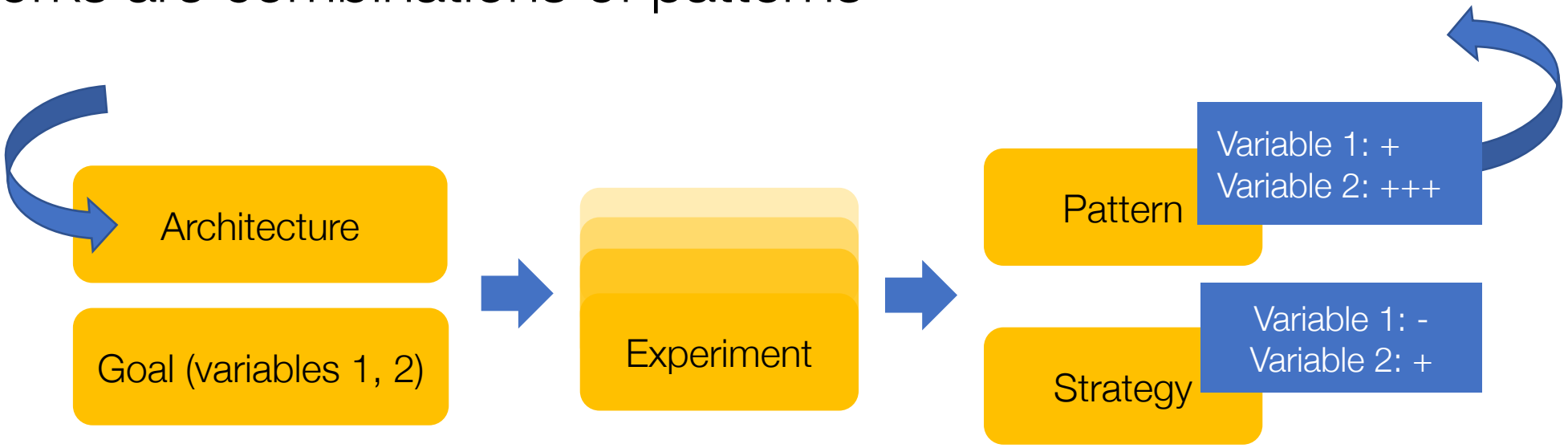
- Continuous assurance
  - Re-design / refactoring at run-time
  - Live experimentation and adaptation
  - The system contributes to its own evolution

Feedback ←



Experimentation for Non-Functional Requirements at run-time

- A given architecture copes with contextual situations differently
- Each goal requires monitoring specific variables
- An experiment is an architecture fork, deployed as a *dark launch*
- Forks are combinations of patterns



Key challenges

1. Specification of patterns (codification)
2. Specification and instrumentation of metrics for each pattern
3. Specification of goals / constraints regarding non-functional requirements

Quantifiable relationship between patterns and requirements.
Is a description enough for engineering?

Component model for microservices (deployment) Future work

1. Specification of patterns (codification)

- Patterns differ in nature: software, network, deployment, …
- Extend Architecture Description Languages (ADLs)
  - Given an architecture
    - Recognize applied patterns
    - Apply a pattern
  - Operationalize ADLs
    - E.g., adjust network bandwidth, unbind/bind services, create skeletons

2. Specification and instrumentation of metrics

- It’s not only about monitoring but getting specific variables
  → requires data gathering, aggregation and filtering

- Approaches:

  Policies
  Comprehensible set of variables and procedures to monitor them

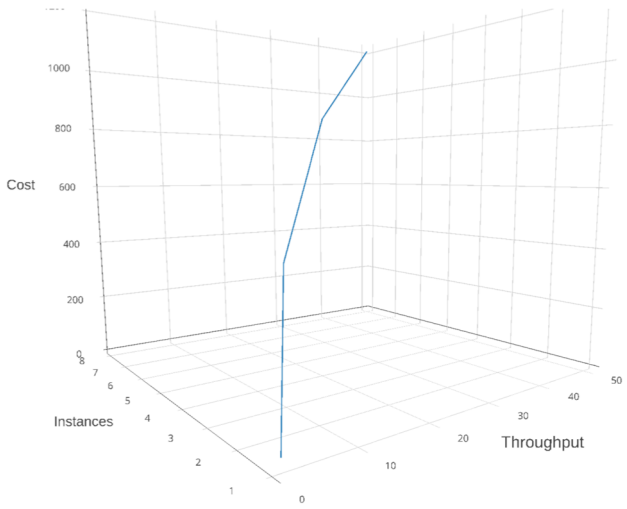
  Monitors
  DSL to specify monitors using introspection → likely requires component model

  Hybrid
  Declarative policies based on a collection of monitor specifications

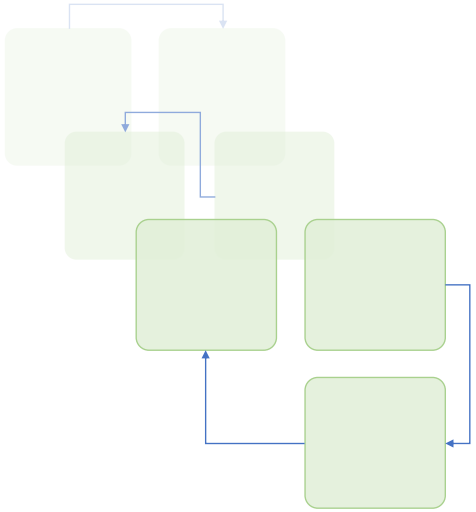
3. Specification of goals and constraints

- Based on the architecture and the monitoring variables, compose complex constraints
- Machine and human readable goals / constraints
- Goals models are an alternative. Extending the notation may be necessary

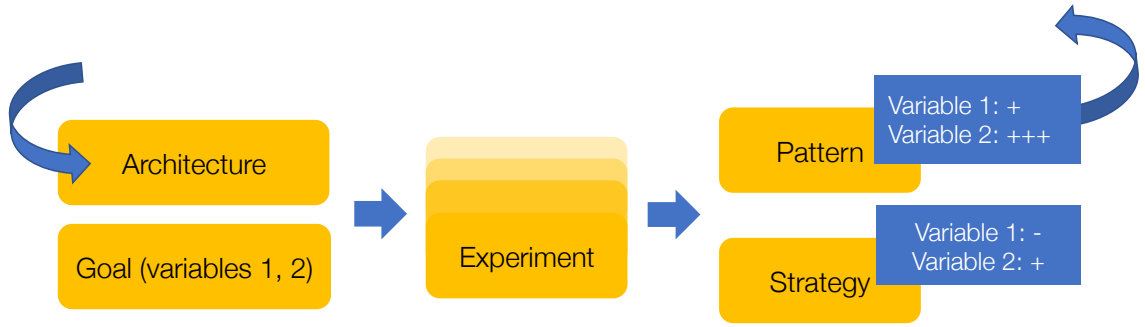
Recap



Continuous assurance



Architecture forks



Controlled Experiments

Thanks!